System Architecture Specification

(*Architekturspezifikation*)

(TINF18C, SWE I Praxisprojekt 2018/2019)

Project: AMLEngine-DLL Interfaces

Customer: Rentschler & Ewertz
Rotebühlplatz 41
70178 Stuttgart

Supplier: Team 4 Joshua, Kevin, Krister, Lucas, Markus, Robin
Rotebühlplatz 41
70178 Stuttgart

Version	Date	Author	Comment
0.1	21.10.2019	Robin	Document created
0.2	25.10.2019	Robin	Added System overview & Architectural concepts
0.3	04.11.2019	Robin	Finished Subsystem Specifications, System Design, Technical Concepts
1.0	06.11.2019	Robin	Fixed last details
1.5	25.04.2020	Lucas	Updated
2.0	03.05.2020	Lucas	Completed

Table of Content

1. Introduction

The goal of this project is to develop software which facilitates the usage of the AMLEngine Version 2.1 [1] in the programming languages C++ and JavaScript.
In the case of JavaScript to achieve this, the functionality shall be provided in the form of code, which can be used in other projects to interface with the AML.Engine.dll.
For C++ however, the wrapper is a collection of instructions and examples on how to use the AML.Engine.dll, not actual code with which other software interfaces.
The application also provides a console application tool, which validates and (de)compresses AMLX files.

2. System Overview

2.1. System environment

The different parts of the system are used in several environments. The wrappers are used in software projects by developers to access the functionality of the AML.Engine.dll. The console application on the other hand is run in an environment where a user wants to use it together with an AML file.

2.2. Software environment

The C++ wrapper instructions shall explain how to use the AML.Engine.dll in C++ projects running on devices with the .NET Framework version 4.7 or 4.8 installed. It explains the usage with the Microsoft C++ Compiler.
The JavaScript wrapper shall allow the usage of the AML.Engine.dll in Node.js [2] projects targeting Node.js version 12.x.x LTS on devices with access to .NET Framework version 4.7 or 4.8.
The console application shall be usable on devices on which .NET Framework 4.7 or 4.8 is available.
Furthermore, both wrappers as well as the console application require the already imported Aml.Engine.dll in version 1.3.6 against which they have been developed. Newer versions may be supported but are not guaranteed to do so.

2.3. Hardware Environment

The project runs on a common client PC or servers which run Windows 7 or later.

3. Architectural Concept

This project can be divided into three main parts: the C++ wrapper, the JavaScript wrapper and the console application.

The C++ wrapper is not actual code but a list of instructions on how to compile C++ code which uses the AML.Engine.dll. Apart from instructions, there is also an example project code example. These instructions are available for the Microsoft C++ Compiler. They consist of a list of compiler settings. Usage examples will also be provided. They start from installing all necessary dependencies.

The JavaScript wrapper is source code which allows to use functionality from the AML.Engine.dll in the JavaScript runtime environment Node.js. Therefore, it can invoke methods from the AML.Engine.dll. Additionally, a small number of examples will be provided. This source code can easily integrate in one's project with the Node package manager [3].

The console application includes a module for validating an AML file. This module uses functions included in the AMLEngine to analyse the file and issue an error message that contains the reason for the error and the line number on which the error occurred. The console application will also include another module to extract all the files contained in an AMLX file and compress the files back to an AMLX file.

3.1. Quality Goals

The following points describe the goals that should be achieved by the components of this project.

3.1.1. Usability

In order to provide optimal usability, the goal is to create detailed documentation. This also includes the creation of example code with inline comments to demonstrate the usage of the wrappers.

3.1.2. Maintainability

In order to provide optimal maintainability, the console application will be divided into different modules according to object-oriented programming principles.

The JavaScript wrapper will also be separated into modules to make it easy to analyse and modify.

3.1.3. Efficiency

Another focus in the project is, to keep it simple and efficient. Therefore, the goal is to introduce as little overhead as possible for the wrappers.

3.1.4. Portability

The console application shall be usable without requiring any installation and available as a simple executable file. Therefore, it will be fully portable. This point is not applicable to the wrappers.

3.2. Architectural Model

The console application consists of three layers. The first one is the AML.Engine.dll itself. In this part, the business logic will be done. The second layer is the logic of the console application. This layer should manage the parameters and the function calls for the DLL. The last layer is the console interface. This is the layer that the user can directly access and from which he can control the application.

The JavaScript wrapper consists of three layers as well. The first one is the same as in the console application. It’s the AML.Enigne.dll. The second one, is the wrapper which will be addressed by the Edge Node.js module. This wrapper forwards the function calls in a valid style to the DLL and connects the node project with the DLL. The last one is the node-project. Here the developer can access the wrapper and use function calls to the wrapper to use the DLL and manage the AML file.

The C++ wrapper documentation has no architectural model, because it is only a documentation and not a software project. The structure of the wrapper document has been separated into four parts. It starts with the installation of the environment. In this environment the dependencies will be installed in the part two. After this, some sample code will be added. In the end, the compiler with the correct settings can be started.

4. System Design

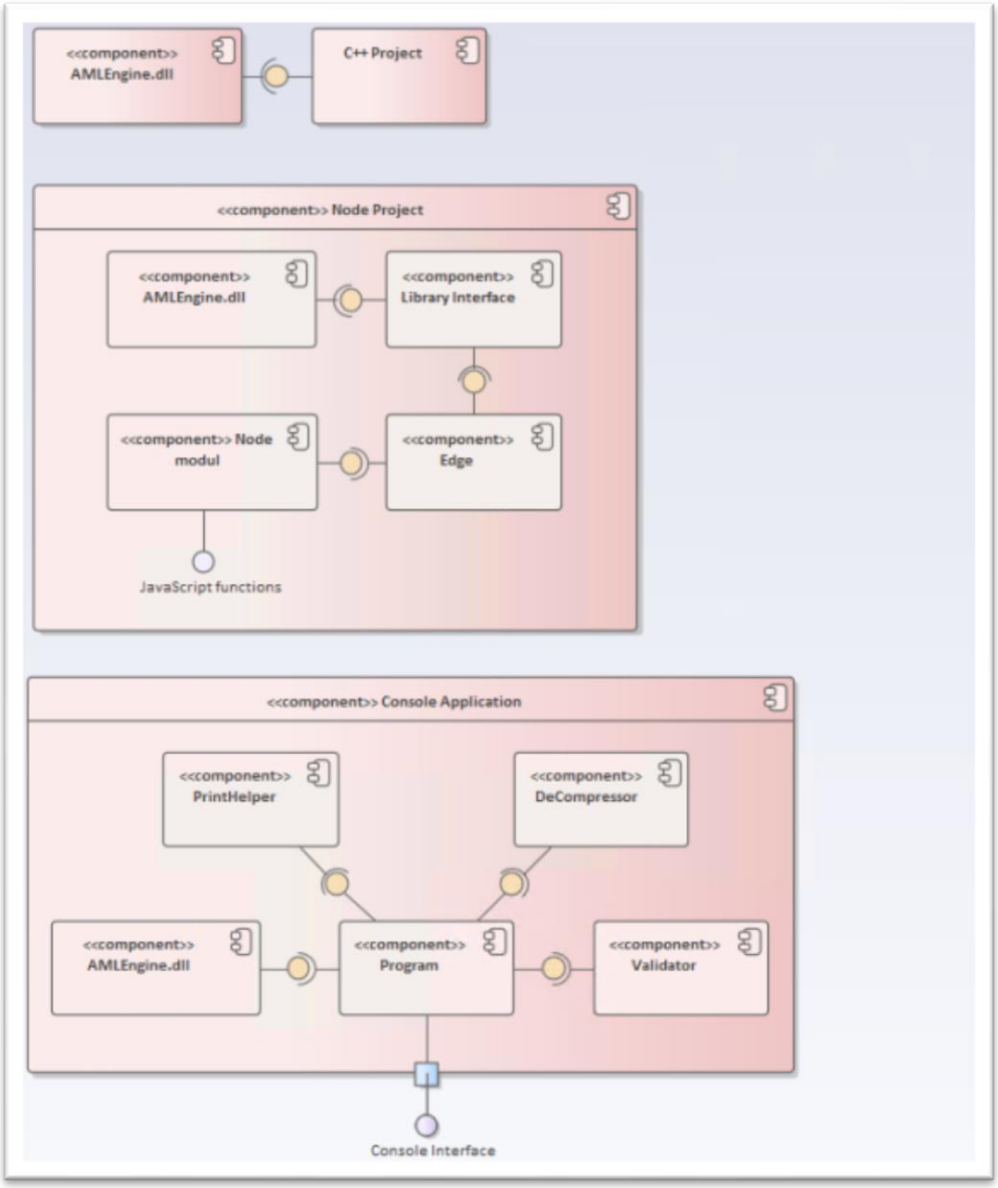

Figure 1: Component Diagram

5. Subsystem Specifications

5.1. <MOD.010>: C++ Wrapper

<MOD.010>	C++ wrapper
System requirements covered	LF10, LF20
Service	Supporting the developer to compile their code
Interfaces	
External Data	● AML.Engine.dll ● AML File
Storage Location	Not applicable
Open Points	

5.2. <MOD.020>: JS Wrapper

<MOD.020>	JS wrapper
System requirements covered	LF30, LF40
Service	Enabling the developer to use the functions of the AML.Engine.dll in their Node Project.
Interfaces	JavaScript functions which provide the same functionalities as the AMLEngine.
External Data	● AML.Engine.dll ● AML file ● Edge module
Storage Location	In a windows directory
Open Points	

5.3. <MOD.030>: Console Application

<MOD.030>	Console Application
System requirements covered	LF50, LF60, LF70

Service	Loading of AML files and AMLX files Validation and (un-)packing of AML /AMLX files
Interfaces	Windows Command Prompt
External Data	● AML.Engine.dll ● AML file ● AMLX file
Storage Location	In a windows directory
Open Points	

5.4. <MOD.031>: Console Application - Program class

<MOD.031>	Console Application: Program class
System requirements covered	LF50
Service	Loading of AML and AMLX files
Interfaces	Windows Command Prompt
External Data	● AML.Engine.dll ● AML file ● AMLX file
Storage Location	Specified in <MOD.030>
Open Points	

5.5. <MOD.032>: Console Application - Validator class

<MOD.032>	Console Application - Validator class
System requirements covered	LF70
Service	Validates AML files
Interfaces	Function to validate CAEXDocuments and print error messages
External Data	● CAEXDocuments stored in memory
Storage Location	Specified in <MOD.030>
Open Points	

5.6. <MOD.033>: Console Application - (De-)Compressor

<MOD.033>	Console Application - (De-)Compressor
System requirements covered	LF60
Service	Enabling the user to compress, decompress, load and store AML files or AMLX files
Interfaces	Function to compress or decompress files.
External Data	● AML.Engine.dll ● AML files ● AMLX file
Storage Location	Specified in <MOD.030>
Open Points	

6. Technical Concepts

6.1. Persistence

Only applicable to the console application. Files shall be compressed and stored as an AMLX file in the location of the users choosing.

6.2. User Interface

The user interface for the console application is specified in the <MOD-030>.

6.3. Ergonomy

Not applicable for this project.

6.4. Transaction Control

Not applicable for this project.

6.5. Session Control

Not applicable for this project.

6.6. Communication with other IT-Systems

For the wrappers the communication will be handled by the developer, who uses them.
The console application has no communication with other IT-Systems.

6.7. Deployment

The wrappers will be imported by developers either through a copy of the code or through the packaging system Node package Manager.
The console application executable needs to be downloaded to the user's device.

6.8. Data Validation

The data validation is specified in the <MOD-032>.

6.9. Exception Handling

For the JavaScript wrapper errors will be passed through to the developer and not handled by the wrapper itself.
The console application will throw exceptions when the file is invalid. Further specified in <MOD-030>.

6.10. Logging

Not applicable for this project.

6.11. Configurability

Not applicable for the wrappers.
Configurability will be handled through parameters in the console application.

6.12. Parallelisation

Not applicable for this project.

6.13. Internationalisation

All documentation, comments and the console application will be in English. Therefore, it is accessible to a wide user base.

6.14. Migration

Not applicable for this project, because there is no old system of this project.

6.15. Testability

The project will be tested with unit as well as user tests.

6.16. Scalability

Not applicable for this project.

6.17. Availability

Not applicable for this project.

7. References

[1] AMLEngine - https://github.com/AutomationML/AMLEngine2.1/
[2] JavaScript runtime environment Node.js - https://nodejs.org/
[3] Node package Manager (NPM) - https://www.npmjs.com/

8. Glossary

.NET	The .NET Framework is a software development and runtime developed by Microsoft for Microsoft Windows.
AML	Automation Mark-up Language is an open standard data format for storing and exchanging plant planning data.
AMLX	Multiple files can be stored compressed in an AML-Container (.amlx).
C++	C++ is an extension of the C programming language.
CAEX	Computer Aided Engineering Exchange
CLI	The Console Application Interface from Microsoft Windows.
DLL	Dynamic Link Library is a file format used to store precompiled code
GUI	Graphical User Interface
JS	JavaScript is a scripting language which supports dynamic typing.
NPM	Node Package Manager
Runtime environment	A runtime environment is an environment provided by the operating system for granting access to other system resources such as RAM.